# Numerical Methods ITGS219

## Lecture Newton–Raphson and Secant Methods

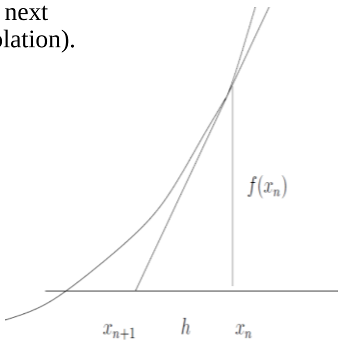
## Root Finding

*By: Zahra A. Elashaal*

## Root Finding by Newton–Raphson

The central premise for both methods is that the function is locally linear and the next iteration for the required value can be attained via linear extrapolation (or interpolation).

**Derivation of the Newton–Raphson Method**



- We will start using a Taylor series to derive the Newton–Raphson technique. We assume the current guess is $x$ and this is *incremented* by an amount *h, so that x + h is the required value.* It now remains for us to determine $h$ or at least find an approximation to it.
- The Taylor expansion for the function *f(x)* at the point $x + h$ *is given by:*

$$f(x+h) = f(x) + hf'(x) + O(h^2).$$

- This can be interpreted as: the value of the function at $x + h$ *is equal to the* value of the function at $x$ plus the gradient times the distance between the points. This can be considered to include further terms; at the moment we are fitting a straight line.
- We now note that $x + h$ is supposedly the actual root so $f(x+h) = 0$, and discarding the higher-order terms $O(h^2)$ we find that

$$h \approx -\frac{f(x)}{f'(x)}.$$

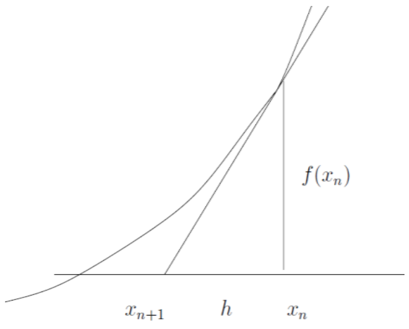
## Root Finding (Newton–Raphson)

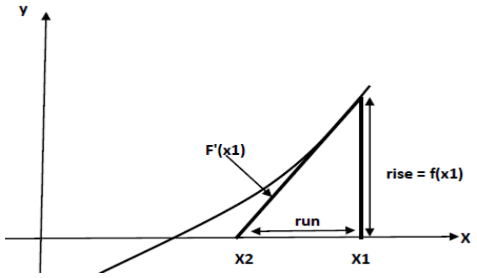
This presumes we are close to the actual root, and consequently we can discard the terms proportional to $h^2$ since these should be smaller than those proportional to $h$.

- This allows us to construct the iterative scheme

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, 2, \cdots .$$

- This method can also be derived using geometric arguments. In these derivations the function is taken to be approximated by a straight line in order to determine the next point.





F'(x1) = rise / run

F'(x1) = f(x1)/run

Run = f(x1)/f'(x1)

X1-x2 = f(x1)/f'(x1)

X2= x1-(f(x1)/f'(x1)) ----- newton's formula.

## Root Finding (Newton–Raphson)

At the moment we shall presume that we have two routines *func.m* and *func_prime.m* which give us the function and its derivative.

- For ease let us consider the function $f(x) = x - 2\sin x^2$, so $f'(x) = 1 - 4x\cos x^2$ (using the chain rule). The code *func.m* and *func_prime.m* is given as:

```
% func.m
function [f] = func(x)
f = x-2*sin(x.^2);
```

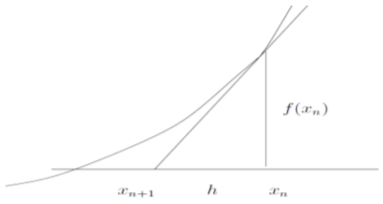
```
% func_prime.m
function [value] = func_prime(x)
value = 1 - 4*x.*cos(x.^2);
```

- Notice that we have used the dot operators, even though this routine is only ever likely to be called in this context using a scalar. This permits the routine to be used from other codes in a portable fashion.
- This can be coded simply using

```
x = 1;
for j = 1:10
     x = x - func(x)/func_prime(x);
end
```

- where we have set the initial guess to be $x = 1$ and supposed that the method will converge in 10 iterations.

## Root Finding (Newton–Raphson)



We now give a more robust code to perform the iterations

- Hopefully you can see the difference between the two codes and see that ultimately
- the second version is more useful. By entering the values 0.1, 0.6 and 1.5 we can obtain the three roots we are concerned with.
- Notice we have increased the number of digits printed in the answer to 10 using the form of the MATLAB command num2str which accepts two arguments.

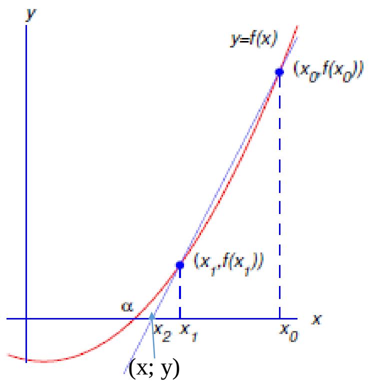
```
% Newton_Raphson.m
x = input('Starting guess: ');
tolerance = 1e-8;
iterations = 0;
while (iterations<30) & (abs(func(x))>tolerance)
    x = x-func(x)/func_prime(x);
    iterations = iterations + 1;
end
if iterations==30
    disp('No root found')
else
  disp([' Root = ' num2str(x,10) ' found in ' ...
   int2str(iterations) ' iterations.'])
end
```

## Root Finding ( Secant and False Position Methods )

We will introduce two additional methods to find numerical solutions of the equation $f(x) = 0$. The **Secant Method** and **False Position Method.** Both of these methods are based on approximating the function by secant lines just as Newton's method was based on approximating the function by tangent lines.

### The Secant Method

The secant method requires two initial approximations $x_0$ and $x_1$, preferably both reasonably close to the solution $x_r$. From $x_0$ and $x_1$ we can determine that the points ($x_0$; $y_0 = f(x_0)$) and ($x_1$; $y_1 = f(x_1)$) both lie on the graph of $f$. Connecting these points gives the (secant line).



$$\frac{y - f(x_1)}{f(x_1) - f(x_0)} = \frac{x - x_1}{x_1 - x_0} \quad \text{or} \quad \frac{y - f(x_0)}{f(x_1) - f(x_0)} = \frac{x - x_0}{x_1 - x_0}$$

- Since we want f(x) = 0, we set y = 0, solve for $x$ ($x_2$ *say*), and use that as our next approximation.

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)} = x_0 - f(x_0)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

- We then move on so that we are going to use $x_0 = x_1$ and $x_1 = x_2$ as the next two points.

Repeating this process gives us the iteration with $y_i = f(x_i)$.

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

## Root Finding( Secant Methods )

Example, suppose f(x) = $x^4$ - 5, which has a solution $x_r = \sqrt[4]{5} \approx 1.5$ Choose $x_0 = 1$ and $x_1 = 2$ as initial approximations.

Next we have that $y_0$ = f(1) = -4 and $y_1$ = f(2) = 11. We may then calculate $x_2$ from the formula:

$$x_2 = x_1 - f(x_1)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_2 = 2 - 11\,\frac{2-1}{11-(-4)} = \frac{19}{15} \approx 1.2666\,\ldots.$$

Pluggin $x_2$ = 19/15 into f(x) we obtain $y_2$ = f(19/15) = -2.425758… .

In the next step we would use $x_1$ = 2 and $x_2$ = 19/15 in the formula to find $x_3$ and so on.

**Exercises**

Perform 3 iterations of the secant method on the function f(x) = $x^3$ - 4, with starting points $x_{-1}$ = 1 and $x_0$ = 3. (By hand, but use a calculator.)
Calculate the errors and percentage errors of x1, x2, and x3.

## Root Finding (Secant Methods)

- We wish to find the value of *x* *($x_2$ say)* for which *y = 0* which is given by using Matlab and from the equation:

$$x_2 = x_1 - f(x_1)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

- We then move on so that we are going to use $x_1$ and $x_2$ as the next two points.

- This is coded to give:

```
% Secant.m
x0 = input('Starting guess point 1 :');
x1 = input('Starting guess point 2 :');
tolerance = 1e-8;
iterations = 0;
while (iterations<30) & (abs(func(x1))>tolerance)
    iterations = iterations + 1 ;
    f0 = func(x0);
    f1 = func(x1);
    x2 = x1-f1*(x1-x0)/(f1-f0);
    x0 = x1;
    x1 = x2;
end
if iterations==30
    disp('No root found')
else
    disp([' Root = ' num2str(x1,10) ' found in ' ...
    num2str(iterations) ' iterations.'])
end
```

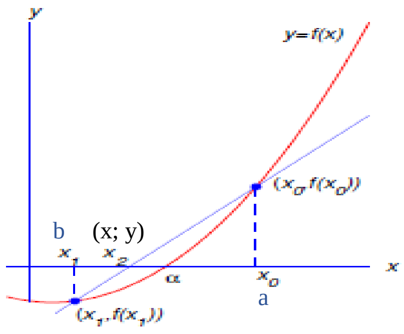
## Root Finding (The *Regula Falsi* (False Position) Method)

### The False Position Method

The False Position method is a combination of the secant method and bisection method. As in the bisection method, we have to start with two approximations a and b for which *f(a)* and *f(b)* have different signs ( *f(a) . f(b) < 0* ). As in the secant method, we follow the secant line to get a new approximation, which gives a formula similar to,

$$x = b - f(b)\frac{b - a}{f(b) - f(a)}$$

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$



- Then, as in the bisection method, we check the sign of f(x); if it is the same as the sign of f(a) then x becomes the new ***a*** and otherwise let x becomes the new ***b***.
- Note that in general either a $\rightarrow x_r$ or b $\rightarrow x_r$ but not both.

## Root Finding (The *Regula Falsi* (False Position) Method)

**Example:** We use the Method of Regula Falsi to solve f(x) = 0 where $f(x) = x^2 - 2$.

First, we must choose two initial guesses $x_0$ and $x_1$ such that f(x) changes sign between $x_0$ and $x_1$. Choosing $x_0 = 1$ and $x_1 = 1.5$, we see that $f(x_0) = f(1) = -1$ and $f(x_1) = f(1.5) = 0.25$, so these choices are suitable.

Next, we use the Secant Method to compute the next iterate $x_2$ by determining the point at which the secant line passing through the points $(x_0; f(x_0))$ and $(x_1; f(x_1))$ intersects the line y = 0. We have

$$x_2 = x_0 - f(x_0)\frac{x_1 - x_0}{f(x_1) - f(x_0)} = 1 - (-1)\frac{1.5 - 1}{0.25 - (-1)} = 1 + \frac{0.5}{1.25} = 1.4$$

Computing $f(x_2) = f(1.4) = -0.04 < 0$. Since $f(x_2) < 0$ and $f(x_1) > 0$, we can replace $x_0$ by $x_2$ to conclude that a solution exists in the interval $(x_2; x_1)$. Therefore, we compute $x_3$ by determining where the secant line through the points $(x_1; f(x_1))$ and $f(x_2; f(x_2))$ intersects the line y = 0. Using the Secant Method, we obtain

$$x_3 = x_1 - f(x_1)\frac{x_2 - x_1}{f(x_2) - f(x_1)} = 1.5 - (0.25)\frac{1.4 - 1.5}{-0.04 - 0.25} = 1.41379$$

We do know that a solution exists in the interval (x3; x1), because f(x1) > 0 and f(x3) < 0 . Therefore, we use the secant line determined by x1 and x3 to compute x4.

## Root Finding (False Position Method)

- This method works far better if the two initial points are on opposite sides of the root.
- In the above method we have merely chosen to proceed to use $x_1$ and the newly attained point $x_2$; however we could equally have chosen $x_0$ and $x_2$.
- In order to determine which we should use we require that the function *changes sign* between the two ends of the interval.
- This is done by changing the lines where the next interval is chosen.

$$x_2 = x_1 - f(x_1)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

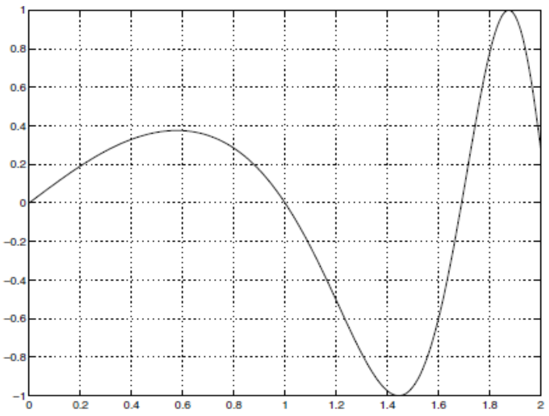
$$x_2 = x_0 - f(x_0)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

```
%False_Position.m
x0 = input('Starting guess point 1 :');
x1 = input('Starting guess point 2 :');
x2 = x0;
tol = 1e-8;
iters = 0;
while ((iters<30) & (abs(func(x2))>tol))|(iters==0)
     iters = iters + 1 ;
     f0 = func(x0);
     f1 = func(x1);
     x2 = x0-f0*(x1-x0)/(f1-f0);
     if func(x2)*f0 < 0
          x1 = x2;
     else
          x0 = x2;
     end
end
if iters==30
     disp('No root found')
else
     disp([' Root = ' num2str(x2,10) ' found in ' ...
     num2str(iters) ' iters.'])
end
```

## Root Finding (The Regula Falsi (False Position) Method)

**Example:** Using the method of False Position find the roots of the function $\sin(x - x^3)$ between x = 0 and x = 6.

Note: The zeros of this function occur where $x - x^3 = n\pi$. In order to obtain initial estimates for the range we plot the function $\sin(x - x^3)$ .



The first root corresponds to n = 0. We note that the function gets very oscillatory as x increases and may show problems as more roots are required, in which case the roots of the cubic $x-x^3 = n\pi$ can be sought, using for instance roots.

Let $x_0 = 1.6$ and $x_1 = 1.8$ and complete the solution to find the root and its iteration.

complete the solution

## Convergence

- If we can begin with a good choice $x_0$, then:
  - Newton's method will converge to $x_r$ rapidly.
  - The secant method is a little slower than Newton's method
  - and the Regula Falsi method is slightly slower than that.

  However, both are still much faster than the bisection method.

- If we <u>do not</u> have a good starting point or interval, then:
  - The secant method, just like Newton's method, can fail altogether.
  - The Regula Falsi method, just like the bisection method, always works because it keeps the solution inside a definite interval.

## Root Finding( Newton–Raphson and Secant Methods )

***Example 4.3*** consider the function *f(x)=* $x^m - a$ where $a>0$. The roots of this equation are $\sqrt[m]{a}$ or, written another way, $a^{1/m}$.

For this equation we can write the Newton–Raphson scheme as:

$$x_{n+1} = x_n - \frac{x_n^m - a}{m x_n^{m-1}},$$

*which can be simplified to yield:*

$$x_{n+1} = x_n \left(1 - \frac{1}{m}\right) + \frac{a}{m x_n^{m-1}}.$$

*Notice this is exactly the map solved in the example on page 91, with a = 3 and m = 2.*

***Example 4.4*** Using the Newton–Raphson routine determine the zero of the *function*

$$f(x) = e^x - e^{-2x} + 1.$$

This can be done by setting up the functions

```
function [value] = func(x)
value = exp(x) -exp(-2*x)+1;
```

```
function [value] = func_prime(x)
value = exp(x)+2*exp(-2*x);
```

*This yields the result*

```
>> Newton_Raphson
Starting guess :-2
Root = -0.2811995743
found in 8 iterations.
```

(with a tolerance of $1\times10^{-10}$). In fact we can solve this equation by introducing the variable $Y = e^x$, and noting that Y is never zero. We can rewrite f(x) as

$$f(x) = \frac{1}{Y^2}(Y^3 - 1 + Y^2)$$

*real root* is found using the Matlab code *co=[1 1 0 -1]; roots(co). Then to retrieve the root of the equation we use the fact that x = lnY.*

## Matlab Routines for Finding Zeros

### Roots of a Polynomial

- As we have seen Matlab has a specific command for finding the roots of a polynomial, namely ***roots***.
- The *coefficients* of the polynomial are placed in a vector ***c*** and the routine returns the corresponding roots.
- It is very simple to use, but care is need when entering the coefficients.

**Example 4.6** Find the roots of the quantic equation $f(x) = x^5 - x^4 + x^3 + 2x^2 - 1$.

This is accomplished using the code:

```
c = [1  -1   1  2  0  -1];
roots(c)
```

There are a couple of things to note from this example:

- The polynomial's coefficients are listed starting with the one corresponding to the **largest power**.
- It is crucial that **zeros** are included in the sequence where necessary (in the above we have included the zero times *x term).*
- As a simple check, a polynomial of **order** or **degree *p*** *has* ***p + 1 coefficients,*** and it *will have* ***p*** **roots**.
- So as long as the coefficients of the polynomial are real, the roots will be real or occur in complex conjugate pairs.

## MATLAB Routines for Finding Zeros ( Cont.)

### The Command *fzero*

The Matlab command *fzero* is very powerful. It actually chooses which scheme should be used to solve a given problem. Let us try the form of the command.

```
fzero ('func', 0.4, optimset('disp','iter'))
```

This produces

The command uses many different forms but here we are looking for a root of the function defined in *func.m* near the value $x = 0.4$ and the *options are set* to display *iterations*.

```
Func-count        x             f(x)            Procedure
    1            0.4         0.0813636           initial
    2       0.388686         0.0876803           search
    3       0.411314         0.0745675           search
    4          0.384         0.0901556           search
    5          0.416          0.071613           search
    6       0.377373         0.0935142           search
    7       0.422627          0.067296           search
    8          0.368         0.0979791           search
    9          0.432         0.0609148           search
   10       0.354745          0.103721           search
   11       0.445255         0.0513434           search
   12          0.336          0.110687           search
   13          0.464         0.0367268           search
   14        0.30949          0.118215           search
   15        0.49051         0.0139394           search
   16          0.272          0.124167           search
   17          0.528        -0.0223736           search

   Looking for a zero in the interval [0.272, 0.528]

   18       0.488914         0.0153797           interpolation
   19       0.504837       0.000616579           interpolation
   20       0.505484       -1.5963e-06           interpolation
   21       0.505482       1.80934e-09           interpolation
   22       0.505482       5.32907e-15           interpolation
   23       0.505482                 0           interpolation
Zero found in the interval: [0.272, 0.528].

ans =

   0.50548227233930
```

## MATLAB Routines for Finding Zeros ( Cont..)

**Example 4.7** Determine the zero of the function ***f(x) = x − x² sin x*** nearest to ***x = 1***.

We need to set up the code

```
function [val] = myfunc(x)
val = x-x.ˆ2.*sin(x);
```

and then use the inline command ***fzero***(’*myfunc*’,*1*). and This gives

```
>> fzero(’myfunc’,1)
       Zero found in the interval: [0.84, 1.16].
       ans  =      1.1142
```

HomeWork tasks Page 130 - 132 from Task 4.2 To Task 4.13

## Any Question?